Latent Regression Bayesian Network for Data Representation

Siqi Nie[*]
Qiang Ji[†]

June 16, 2015

Abstract

Deep directed generative models have attracted much attention recently due to their expressive representation power and the ability of ancestral sampling. One major difficulty of learning directed models with many latent variables is the intractable inference. To address this problem, most existing algorithms make assumptions to render the latent variables independent of each other, either by designing specific priors, or by approximating the true posterior using a factorized distribution. We believe the correlations among latent variables are crucial for faithful data representation. Driven by this idea, we propose an inference method based on the conditional pseudo-likelihood that preserves the dependencies among the latent variables. For learning, we propose to employ the hard Expectation Maximization (EM) algorithm, which avoids the intractability of the traditional EM by max-out instead of sum-out to compute the data likelihood. Qualitative and quantitative evaluations of our model against state of the art deep models on benchmark datasets demonstrate the effectiveness of the proposed algorithm in data representation and reconstruction.

1 Introduction

Deep directed generative models have received increasing attention recently, because the top-down connections explicitly model the data generating process. Different levels of latent variables capture features (or abstractions [15]) in a coarse-to-fine manner. Compared with undirected models such as restricted Boltzmann machines (RBMs) and deep Boltzmann machines (DBMs) [17], directed generative models have their own advantages. First, samples can be easily obtained by straightforward ancestral sampling without the need for Markov chain Monte Carlo (MCMC) methods. Second, there is no partition function issue since the joint distribution is obtained by multiplying all local conditional probabilities, which requires no further normalization. Last but most importantly, the latent variables are dependent on each other given the observations

[*]Email: nies@rpi.edu. Affiliation: Rensselaer Polytechnic Institute, USA.
[†]Email: jiq@rpi.edu. Affiliation: Rensselaer Polytechnic Institute, USA.

through the so-called "explain-away" principle. Through their inter dependency, latent variables coordinate with each other to better explain the patterns in the visible layer.

Learning directed models with many latent variables is challenging, mainly due to the intractable computation of the posterior probability. Although Markov Chain Monte Carlo (MCMC) method is a straightforward solution, the mixing stage is often too slow. To simplify the inference for deep belief networks, Hinton et al. [8] introduced a complementary prior for the latent variables which makes the posterior fully factorized. Some recent efforts for learning generative models have focused on variational methods [12, 13, 16], by introducing another distribution to approximate the true posterior and maximize a variational lower bound of the data likelihood. The approximating distribution is typically fully factorized for computational efficiency. However, the assumption of the factorized distribution sacrifices the " explain-away" effect for efficient inference, which inevitably enlarges the distance to the true posterior, and weakens the representation power of the model. This defeats a major advantage of directed graphical models.

In this work, we address the problem of learning deep directed models in a different direction. We propose to use the EM algorithm with two approximations in the inference and learning phases. First, we approximate the true posterior distribution during inference by the conditional pseudo-likelihood, which preserves to certain degree the dependencies among latent variables. Second, we approximate the data likelihood using a max-out setting during the E-step of the learning to overcome the exponential number of configurations of the latent variables. As a result, the E-step requires the maximum a posteriori (MAP) inference, which is efficiently solved based on the pseudo-likelihood. It can also be seen as the application of iterated conditional modes (ICM) [2] to directed graphical models. In the M-step, the problem is transferred into parameter learning with complete data, which is much easier to handle.

2 Related Work

The research on learning directed model with latent variables can be dated back to 1990s. A standard approach is the Expectation Maximization (EM) algorithm, which maximizes the expected data log-likelihood for parameter learning. EM algorithm and its variants have been used for learning latent mixture of factor analyzers [6], probabilistic latent semantic indexing [10], probabilistic latent semantic analysis [11] and latent Dirichlet allocation (LDA) [3]. Such models have a few latent variables so that the posterior probability of latent variables can be exactly computed in the E-step.

In the case of many latent variables, exact computation of the posterior is intractable because of the exponential number of the latent variable configurations. Patel et al. [15] make one latent variable connecting to a small patch of the input data. Therefore each patch and the corresponding latent variable form a small model, which allows exact maximum a posteriori (MAP) inference. Many other approaches have been proposed to approximate the posterior probability of latent variables along two directions.

One approach is to replace the true posterior distribution with a factorized distribution as an approximation. This approach was first proposed by Saul et al. [20], known as the mean field theory for learning sigmoid belief networks. A fully factorized variational posterior is introduced to approximate the true posterior distribution of latent variables. Recently, Gan et al. [5] extended the mean field method, and proposed a Bayesian approach to learn deep sigmoid belief networks by introducing sparsity-encoraging priors on the model parameters. Alternatively, the posterior distribution can be approximated using a feed-forward network. The wake-sleep algorithm [9] augments the multi-layer belief networks with feed-forward recognition networks. Wake-sleep alternates between updating the model parameters in the wake phase and the recognition network parameters in the sleep phase. Inspired by this idea, many approaches have been proposed recently for learning directed graphical models by maximizing a variational lower bound on the data log-probability. Mnih and Gregor [13] introduced the neural variational inference and learning (NVIL) algorithm for sigmoid belief networks. A feed-forward inference network is used to obtain exact samples from the variational posterior. A neural network is introduced to reduce the variance of the samples. Kingma and Welling [12] proposed the auto-encoding variational Bayes method for continuous latent variables, in which a reparameterization is employed to efficiently generate samples from the Gaussian distribution. Similarly, Rezende et al. [16] propose a stochastic backpropagation algorithm for learning deep generative models with continuous latent variables. Gregor et al. [7] augment the directed model with an encoder, which is also kind of inference network.

Another direction is to make the posterior probability factorized by specifically designing a prior distribution of latent variables. Hinton et al. [8] proposed a complementary prior to ensure a factorized posterior, and proposed a fast learning algorithm for deep belief networks (DBNs), which is basically a hybrid network with a single undirected layer and several directed layers.

In all the above-mentioned methods, the inference typically assumes independency among latent variables due to special prior or factorized approximation in order to accelerate the inference. Because of this assumption, the inference network is not able to capture the correlations among the latent variables. Therefore the approximate posterior might differ significantly from the underlying true posterior. In this work, we intend to preserve the latent variable dependencies for better data representation. We approximate the posterior probability by the conditional pseudo-likelihood, and employ a hard version of the EM algorithm for parameter estimation.

3 Latent Regression Bayesian Network

We propose a generalized directed graphical model, called latent regression Bayesian network (LRBN), as shown in Fig. 1 (a). The latent variables in LRBN are binary, and the visible variables can be continuous or discrete. Each latent variable is connected to all visible variables. We discuss the parameterization of both cases in the sequel. The case of continuous latent variables can be referred to as factor analyzers [6, 21] or deep latent Gaussian models [12, 16].

3.1 Discrete LRBN

For discrete LRBN, both latent and observation variables are discrete. For brevity, we discuss the binary case with observation variables $x \in \mathbb{B}^{n_d}$ and latent variables $h \in \mathbb{B}^{n_h}$. We assume that the latent variables h determine the patterns in data x, therefore directed links are used to model their relationships, as in a Bayesian network.

Prior probability for latent variables is represented as a log-liner model,

$$P(h_j = 1) = \sigma(d_j)\,, \tag{1}$$

where d_j is the parameter defining the prior distribution for node h_j; $\sigma(\cdot)$ is the sigmoid function $\sigma(z) = 1/(1 + e^{-z})$.

The conditional probability given the latent variables is,

$$P(x_i = 1|h) = \sigma(\sum_j w_{ij} h_j + b_i)\,, \tag{2}$$

where w_{ij} is the weight of the link connecting node h_j and x_i; b_i is the offset for node x_i. The joint probability is,

$$P_\theta(x,h) = \frac{\exp\Big(\sum_{i,j} w_{ij} x_i h_j + \sum_i b_i x_i + \sum_j d_j h_j - \sum_i \log\Big(1 + \exp(\sum_j w_{ij} h_j + b_i)\Big)\Big)}{\prod_j (1 + \exp(d_j))}\,. \tag{3}$$

In this case, the model becomes a sigmoid belief network (SBN) [14] with one latent layer. If more layers are added on top, the conditional probability is defined in the same way as Eq. 2. The model is named a regression model based on the nature of the conditional probability. For discrete visible node, the input to the sigmoid function is a linear combination of the latent variables; for continuous visible node, the mean of a visible node is a linear combination of its parent nodes.

As a similar model with undirected links, RBMs (Fig. 1 (b)) have been widely used in the literature for feature learning and data representation. The joint probability defined by a discrete RBM is,

$$P_{\text{RBM}}(x,h) = \frac{1}{Z} \exp\left(\sum_i b_i x_i + \sum_{i,j} w_{ij} x_i h_j + \sum_j d_j h_j\right)\,. \tag{4}$$

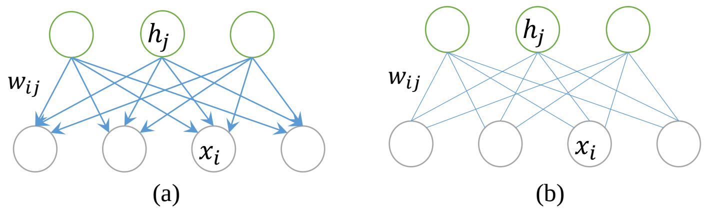


Figure 1: Graph representation of the (a) directed and (b) undirected model for data representation. Each link is associated with a weight parameter.

Comparing Eq. 3 and 4, the additional terms in the numerator captures the correlations among them. This is the reason why $P(h|x)$ is not factorized over individual latent nodes h_j in LRBN, which is the major difference from RBM. An advantage of Eq. 3 is that every term can be computed given the values of all variables, without the issue of the intractable partition function Z.

The discussion of hybrid LRBN is moved to a supplementary material due to the limited space.

3.2 Hybrid LRBN

For hybrid LRBN, the observation variables $x \in \mathbb{R}^{n_d}$ are continuous while the latent variables are binary $h \in \mathbb{B}^{n_h}$. The prior distribution of the latent variables is the same as Eq. 1. Given the latent variables, the visible variable is assumed to follow Gaussian distribution, whose mean is a linear combination of the latent variables,

$$P(x_i|h) \sim \mathcal{N}\left(\sum_j w_{ij}h_j + b_i,\, \sigma_i\right), \quad i = 1, \ldots, n_d\,, \tag{5}$$

where w_{ij} is the weight of the link connecting node h_j and x_i; b_i is the offset of the mean for node x_i; σ_i is the standard deviation. To simplify the learning process, each component of the data is normalized to have zero mean and unit variance, therefore σ_i is set to 1. From the prior distribution and conditional distribution, the joint distribution for x and h is,

$$P_\theta(x, h) = \frac{\exp\left(-\frac{1}{2}||x - Wh - b||^2 + d^T h\right)}{(2\pi)^{n_d/2}\prod_j (1 + \exp(d_j))}\,, \tag{6}$$

or,

$$\begin{aligned} P_\theta(x, h) = &\frac{1}{(2\pi)^{n_d/2}\prod_j (1 + \exp(d_j))} \\ &\exp\left(\frac{1}{2}(x-b)^T(x-b) - x^T Wh + \frac{1}{2}h^T W^T Wh - d^T h\right)\,. \end{aligned} \tag{7}$$

For brevity, vector and matrix forms are used, $W = \{w_{ij}\}$, $b = \{b_i\}$, $d = \{d_j\}$. $\theta = \{W, b, d\}$ represent all the parameters.

For real-valued input data and binary latent variables, Gaussian-Bernoulli RBM defines the joint probability of visible and latent layer,

$$P_{\text{RBM}}(x, h) = \frac{1}{Z}\exp\left(-\frac{1}{2}(x-b)^T(x-b) + x^T Wh + d^T h\right)\,, \tag{8}$$

where Z is the partition function to make $P_{\text{RBM}}(x, h)$ a valid probability distribution. The input data is assumed to be normalized to have unit variance.

Comparing LRBN (Eq. 7) and RBM (Eq. 8), with the same dimensionality of the visible and latent layer, the two models have the same amount of parameters. However, the directed model has a quadratic term $h^T W^T Wh = \sum_i \left(\sum_j w_{ij}h_j\right)^2$, which does not exist in the joint distribution of RBM. This term explicitly captures the correlations among the latent variables h. It also explains why given the visible layer, the latent variables are dependent on each other.

4 LRBN Inference

In this section, we introduce an efficient inference method for LRBN based on conditional pseudo-likelihood. Given a LRBN model with known parameters, the goal of inference is to compute the posterior probability of the latent variables given input data, i.e., computing $P(h|x)$.

In this work, we are interested in the maximum a posteriori (MAP) inference, which is to find the configuration of latent variables that maximizes the posterior probability given observations,

$$h^* = \arg\max_h P(h|x)\,. \tag{9}$$

The MAP inference is motivated by the observation that from the data generating point of view, the variables in one latent layer take values according to the conditional probability given its upper layer. Therefore this configuration dominates all the others in explaining each data sample. In addition, the goal of feature learning is to learn a feature h that best explains x. In this regard, we only care about the most probable states of the latent variables given the observation.

Because of the dependencies among elements of h, direct computing $P(h|x)$ is computationally intractable, in particular when the dimension of h is high. According to the chain rule, the posterior probability of h is,

$$P(h|x) = \prod_j P(h_j|h_1, \ldots, h_{j-1}, x)\,, \tag{10}$$

The pseudo-likelihood replaces the conditional likelihood by a more tractable objective, i.e.,

$$P(h|x) \approx \prod_j P(h_j|h_{-j}, x)\,, \tag{11}$$

where $h_{-j} = \{h_1, \ldots, h_{j-1}, h_{j+1}, \ldots, h_{n_h}\}$ is the set of all latent variables except h_j. In this approximation, we add conditioning over additional variables.

The conditional pseudo-likelihood can be factorized into local conditional probabilities, which can be estimated in parallel. To optimize over the pseudo-likelihood, one latent variable is updated by fixing all other variables,

$$h_j^{t+1} = \arg\max_{h_j} P(h_j|x, h_{-j}^t)\,, \quad 1 \le j \le n_h\,, \tag{12}$$

where t denotes the t^{th} iteration.

Theorem 1 *The updating rule (Eq. 12) guarantees that the posterior probability $P(h|x)$ will only increase or stay the same after each iteration.*

$$P(h_j^{t+1}, h_{-j}^t|x) \ge P(h^t|x)\,. \tag{13}$$

The conditional probability of one latent variable by fixing all other variables is easy to compute, since it involves the computation of the joint distribution twice.

$$P(h_j|x, h_{-j}) = \frac{P(h_j, x, h_{-j})}{\sum_{h'_j} P(h'_j, x, h_{-j})} . \tag{14}$$

In general, computing the joint probability $P(x, h)$ has complexity $O(n_d n_h)$. If each latent variable is updated t times to get h^*, the overall complexity for the LRBN inference is $O(tn_d n_h^2)$, which is much lower than the $O(2^{n_h} n_d n_h)$ complexity when computing $P(h|x)$ directly.

The updating method can be seen as a coordinate ascent algorithm or the iterated conditional modes (ICM) as in inference of Markov random fields. Typically in MRF the number of neighbors for one node is limited. In the case of LRBN, one latent variable is related to all the other variables, due to the rich dependencies encoded in the structure. As discussed above, existing methods to address the inference intractability problem makes the posterior probability completely factorized, therefore sacrificing the dependencies among the latent variables. In contrast, through pseudo-likelihood, we can preserve the dependencies to certain degree.

The inference method requires an initialization for the hidden variables. Different initializations will end up with different local optimal points. To obtain consistent initialization, we drop the direction of the links, and treat the directed model as an undirected one. Therefore, the latent variables are independent of each other given the observations. Specifically, for binary input,

$$P(h_j = 1|x) = \sigma(\sum_i w_{ij} x_i + d_j) . \tag{15}$$

For continuous input, based on Eq. 7, we drop the off-diagonal terms of matrix $W^T W$ for the sake of efficiency, resulting in a factorized distribution of the latent variables,

$$P(h_j = 1|x) = \sigma(\sum_i w_{ij} x_i + d_j - s_j) , \tag{16}$$

where $(s_1, \ldots, s_{n_h}) = diag(\frac{1}{2} W^T W)$. For the initialization, the dependency among the latent variables is ignored, and then through coordinate ascent, it is recovered by updating a subset of variables with others fixed.

5 LRBN Learning

In this section, we introduce an efficient LRBN learning method based on the hard Expectation Maximization (EM) algorithm. The conventional EM algorithm is not an option here due to the intractability of computing posterior probability in the E-step. The hard version of EM algorithm has been explored in [22] for learning a deep Gaussian mixture model. This model has a deep structure in terms of linear transformations, but only has two layers of variables.

5.1 Learning One Latent Layer

Consider the model $P_\theta(x, h)$ defined in section 3. The goal of parameter learning is to estimate the parameters $\theta = \{w, b, d\}$ given a set of data samples $D = \{x^{(m)}\}_{m=1}^{M}$. The conventional maximum likelihood (ML) parameter estimation is to maximize the following objective function,

$$\theta^* = \arg\max_\theta \sum_m \log \sum_h P_\theta(x^{(m)}, h) \,. \tag{17}$$

The second summation in Eq. 17 is intractable due to the exponentially many configurations of h. In this work, we employ a max-out estimation of the data log-probability, with the following objective function,

$$\theta^* = \arg\max_\theta \sum_m \log \max_h P_\theta(x^{(m)}, h) \,. \tag{18}$$

Note that the max-out approximation of the data likelihood is not equivalent to approximating $P(h|x)$ with a delta function. A delta function must be avoided since it is also factorized, which defeats our motivation of preserving the dependency.

With objective function Eq. 18, the learning method becomes a hard version of the EM algorithm, which iteratively fills in the latent variables and update the parameters. In the E-step, $h^* = \arg\max_h P(x, h)$ is effectively estimated using the proposed inference method. In the M-step, the problem of parameter estimation is straightforward because now we are dealing with complete data.

For binary observations, the parameter learning can be decomposed into learning local CPD for each variable, by solving multiple logistic regression problems. The gradient of the parameters is,

$$\frac{\partial \log P(x, h)}{\partial w_{ij}} = h_j \left(x_i - P(x_i = 1|h)\right) \,, \tag{19}$$

$$\frac{\partial \log P(x, h)}{\partial b_i} = x_i - P(x_i = 1|h) \,. \tag{20}$$

In hybrid LRBN, the objective function is convex, and the maximization likelihood solution can be obtained by setting,

$$\sum_m \frac{\partial \log P(x^{(m)}, h^{(m)})}{\partial \theta} = 0 \,. \tag{21}$$

The solution of parameters has a closed form,

$$W = \left(\sum_m (x^{(m)} - b)(h^{(m)})^T\right) \left(\sum_m h^{(m)}(h^{(m)})^T\right)^{-1} \,. \tag{22}$$

In case of large datasets, it is time consuming because all the training instances are used to compute the gradient. Stochastic gradient ascent algorithm can be used to address this issue. The true gradient is approximated by the gradient at a minibatch of training samples. As the algorithm sweeps through the entire training set, it performs the gradient updating for each training sample. Several passes is made over the training set until the algorithm converges. The learning method is summarized in Algorithm 1.

Algorithm 1 Unsupervised Parameter learning of an LRBN with one latent layer.

```
Input training data $X = \{x^{(m)}\}$
Output parameters $\theta$ of an LRBN
1: Initial parameters $\theta$;
2: Initialize the states $h_0$ for all the latent variables using some feed-forward model
   (Section 4);
3: while parameters not converging, do
4:     Random select a minibatch of data instances $x \in X$
5:     Update the corresponding $h$ for $x$ by maximizing the posterior probability, using
       current parameters,
           $h^* = \arg\max_h P_\theta(x, h)\,.$     (23)
6:     Compute the gradients using Eq. 19 and 20. Update the parameters,
           $\theta = \theta + \lambda \nabla_\theta \log P(x, h^*)$     (24)
7: end while
```

5.2 Learning Deep Layers

The learning method of a two-layer LRBN not only provides the parameter θ, but also perform the MAP inference to obtain $h^* = \arg\max_h P(h|x)$. If we denote the features as h^1 and treat them as the input to another LRBN, the same learning procedure can be repeated to learn another layer of features h^2. By doing this we stack another LRBN on top of the first one to build a deep model.

In general, let h^l denote the variables in the l^{th} latent layer ($0 \le l \le L, h^0 = x$), and θ^l be the parameters involved between layer l and $l+1$.

The parameter θ^{l*} is estimated as,

$$\theta^{l*} = \arg\max_{\theta^l} \sum_m \log \max_{h^{l+1}} P_\theta(h^{l,(m)}, h^{l+1})\,, \quad 1 \le l \le L\,. \tag{25}$$

To optimize the objective function, we use the stochastic gradient ascent method, and replace the input X by h^l in Algorithm 1. By performing the layer-wise learning procedure from the first latent layer to the L^{th}, we learn a deep model from bottom-up sequentially. Each time the MAP estimation of one latent layer is treated as input to the next two-layer model. For data instance $x^{(m)}$,

$$h^{l,(m)} = \arg\max_{h^l} P(h^l | h^{l-1,(m)})\,, \quad 1 \le l \le L\,, \tag{26}$$

where $h^{0,(m)} = x^{(m)}$. The layer-wise pre-training procedure extracts different levels of features from the input data, and also provides an initial estimation of the parameters.

5.3 Fine Tuning

The layer-wise training ignores the parameters of other layers when training a model for each layer from bottom-up. To improve the model performance globally, we employ a fine tuning procedure from top-down after the layer-wise pre-training phase. Depending on whether the labels are available or not, the fine tuning can be done in either supervised or unsupervised manner as discussed below.

5.3.1 Unsupervised Fine Tuning

By extending Eq. 18, the objective function for learning with multiple latent layers is,

$$\theta^* = \arg\max_\theta \sum_m \log \max_{\{h^l\}} P_\theta(x^{(m)}, h^1, \ldots, h^L) \,. \tag{27}$$

Given the states in layer l, the variables in layer $l-1$ is independent of the variables in layer $l+1$. Therefore, the unsupervised fine-tuning is performed for every three consecutive layers. The variables in the middle layer is updated with its upper and lower layers fixed,

$$h^{l*} = \arg\max_{h^l} P(h^l | h^{l-1}, h^{l+1}) \,, \quad 1 \le l \le L-1 \,. \tag{28}$$

The conditional probability $P(h^l|h^{l-1}, h^{l+1})$ is also approximated by the conditional pseudo-likelihood,

$$P(h^l|h^{l-1}, h^{l+1}) \approx \prod_j P(h^l_j | h^l_{-j}, h^{l-1}, h^{l+1}) \,. \tag{29}$$

MAP inference is performed by updating one variable with all others fixed,

$$h_j^{t+1} = \arg\max_{h_j} P(h^l_j | h^l_{-j}, h^{l-1}, h^{l+1}) \,, \quad 1 \le j \le n_h \,. \tag{30}$$

To be consistent with bottom-up training, the initialization of the latent variable h^l always follows Eq. 15 and 16.

With the updated layer h^l, we are able to update the parameters θ^{l-1} and θ^l through,

$$\theta^{l-1*} = \arg\max_\theta \sum_m \log P(h^{l-1,(m)}, h^{l,(m)}) \,. \tag{31}$$

and

$$\theta^{l*} = \arg\max_\theta \sum_m \log P(h^{l,(m)}, h^{l+1,(m)}) \,. \tag{32}$$

This process alternates between parameters updating and latent states updating. Therefore, the information is able to propagate among different layers, and the overall quality of the model will increase. The fine tuning proceeds in a top-down manner starting from layers $\{L-2, L-1, L\}$ and ending with layers $\{0, 1, 2\}$. The bottom-up pre-training and top-down fine-tuning procedures are performed iteratively until the parameters converge.

5.3.2 Supervised Fine Tuning

The parameter of the model can be fine-tuned discriminatively if the label information is available. Define a set of target variables $t = (t_1, ..., t_C)$, with C being the total number of classes. $t_c = 1$ if a sample belongs to class c, and $t_k = 0, \forall k \neq c$. The supervised fine-tuning contains three steps. First, a layer-wise pre-training is performed with $L-1$ latent layers (excluding the top layer) using the method discussed in Section 5.2, obtaining θ^l and $h^l, 1 \leq l \leq L-1$.

Second, the parameter θ^L for the top two layers is estimated as,

$$\theta^{L-1*} = \arg\max_{\theta} \sum_{m} \log P(h^{L-1,(m)}, t^{(m)}) \, . \tag{33}$$

This step works with complete data, which does not require inference because t is always observed. Thees two steps form the bottom-up pre-training procedure.

Third, the top-down fine tuning starts with layers $\{L-2, L-1, L\}$ with $h^L = t$ and ends with layers $\{0, 1, 2\}$. Latent states updating follows Eq. 28, and parameter updating follows Eq. 31 and 32.

6 Experiments

In this section, we evaluate the performance of LRBN and compare against other methods on three binary datasets: MNIST, Caltech 101 Silhouettes and OCR letters. Binary datasets are chosen to compare with other models. The extension to real-value datasets is straightforward. The experiments will evaluate representation and reconstruction power of the proposed model.

6.1 Experimental protocol

We trained the LRBN model using stochastic gradient ascent algorithm with learning rate 0.25. The size of the minibatches is set to 20. Two different structures are studied: one hidden layer with 200 variables, and two hidden layers with each layer containing 200 variables, consistent with the configurations in [5, 13]. For each dataset, we randomly selected 100 samples from the training set to form a validation set. The joint probability on the validation set is a criterion for early stopping.

In this section, we first evaluate the MAP configuration of the latent variables through reconstruction. Reconstruction is performed as follows: given a data vector x, perform a MAP inference to get $h^* = \arg\max_h P(h|x)$. Then $\tilde{x} = \arg\max_x P(x|h^*)$ is the reconstructed data. The reconstruction error $|\tilde{x} - x|^2$ can evaluate how well the model fits the data.

The second criterion is the widely used test data log-probability. Directly computing probability $P(x)$ is intractable due to the exponentially many terms in the summation $P(x) = \sum_h P(x, h)$. In this work, we estimate the log-probability using the conservative sampling-based log-likelihood (CSL) method [1],

$$\log \hat{P}(x) = \log \text{mean}_{h \in S} P(x|h) \, , \tag{34}$$

where S is a set of samples h of the latent variables collected from a Markov chain. The expectation of the estimator is a lower bound on the true log-likelihood [1]. Because of the nature of directed models, samples can be collected from the ancestral sampling procedure. Specifically, the top layer is sampled from the prior probability $P(h^L)$, and the lower layers are sampled from the conditional probabilities $P(h^{l-1}|h^l)$. One million samples are used to reach convergence of the estimation of log-probability, and the average of ten repetitions is reported.

The reconstruction error evaluates the quality of the most probable explanation of the latent variables given observations, while the data log-probability evaluates the overall quality of all configurations of latent variables. They are two complementary criteria for model evaluation.

In the experiment, we compare with published results if they are available. For reconstruction we implement the NVIL, RBM, DBN and DBM models following [4, 8, 13, 17] (denoted by (*) in the following tables). Similar log-likelihood achieved by our implementation indicates the correctness of the implementation.

6.2 MNIST dataset

The first experiment is performed on the binary version of the MNIST dataset (thresholding at 0.5). The dataset consists of 70,000 handwritten digits with dimension 28×28. It is partitioned into a training set of 60,000 images and a testing set of 10,000 images.

The average reconstruction errors of different learning models are reported in Table 1. The MAP inference of neural variational inference and learning (NVIL) [13] is through the inference network. For deep belief network (DBN) [19] and deep Boltzmann machine (DBM) [17], the posterior $P(h|x)$ is already factorized, so that the inference is performed individually for each latent variable. The average reconstruction error of the proposed model is 4.56 pixels, which significantly outperforms the other competing methods. This is consistent with our objective function, indicating the most probable explanation contains most information in the input data, which is effectively captured in the proposed model. Some examples of the reconstruction are shown in Fig. 2.

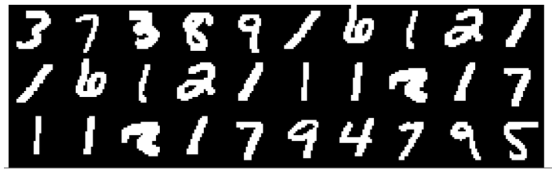
Table 1: Average reconstruction errors of different methods on MNIST dataset. (*) represents our own implementation, same for the following tables.

Method	DIM	Recon Error
NVIL* [13]	200 - 200	35.52
DBN* [19]	200 - 200	29.78
DBM* [17]	200 - 200	23.52
LRBN	200 - 200	4.56

In Table 2 we report the average log-probability on the test set. With the same dimensionality, LRBN outperforms variational Bayes [5], and is similar to that learned using NVIL [13]. Even though our objective function does not explicitly maximize the data likelihood, the learned model achieves comparable performance compared with state of the art learning methods, which indicates that the proposed method is also

Table 2: Test data log-probabilities of different models using on MNIST dataset.

Method	DIM	10k
VB [5]	200	-116.91
VB [5]	200 - 200	-110.74
NVIL [13]	200	-113.1
NVIL [13]	200 - 200	-99.8
DBN [19]	500 - 2000	-86.22
DBM [17]	500 - 1000	-84.62
LRBN	200	-108.7
LRBN	200 - 200	-100.3


(a)

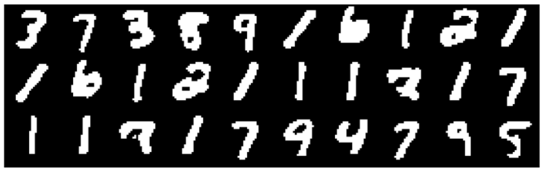

(b)

Figure 2: Examples of the reconstruction. (a) Original digit images. (b) reconstructed by the proposed model.

effective in capturing the distribution of the training data. In all algorithms, introducing a second hidden layer improves the performance. Our method achieves almost 8 nats improvement when additional latent layer is used. Some samples from the generative model are given in Fig. 3.

There is still a gap between the proposed learning method and DBN or DBM. One reason is that we do not have as many latent nodes as in DBN and DBM. Moreover, the max-out approximation of the data likelihood during learning drops all the non-dominant configurations of the latent variables. Therefore it does not necessarily perform well on the task of likelihood comparison. However, it still achieves comparable or even better performance compared to other learning methods.

6.3 Caltech 101 Silhouettes dataset

The second experiment is performed on the Caltech 101 Silhouettes dataset. The dataset contains 6364 training images and 2307 testing images. The reconstruction error is reported in Table 3. The proposed learning method outperforms all the competing methods by a large margin, indicating the effectiveness of the max-out approximation.

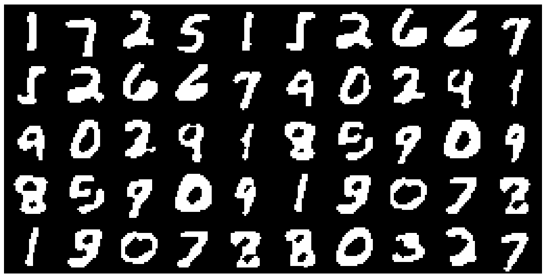


Figure 3: Random samples from the generative model on MNIST dataset.

The test data log-probability is reported in Table 4. With the same dimensionality, the model learned by the proposed algorithm outperforms the one learned by variational Bayes [5], which is considered as one of the state of the art methods of training sigmoid belief networks. With one hidden layer of size 200, the improvement is 34 nats; with a second hidden layer of size 200, the improvement is 28 nats. Moreover, compared to an RBM with much more parameters, our model also achieves better performance, indicating the importance of the underlying dependency of the latent variables. Examples are shown in Fig. 4.

Table 3: Average reconstruction errors of different methods on Caltech 101 Silhouettes dataset.

Method	DIM	Recon Error
NVIL* [13]	200 - 200	29.78
RBM* [4]	200	32.47
DBN* [19]	200 - 200	28.17
DBM* [17]	200 - 200	24.90
LRBN	200 - 200	5.95

Table 4: Test data log-probability.

Method	DIM	Log-prob
VB [5]	200	-136.84
VB [5]	200 - 200	-125.60
RBM [4]	4000	-107.78
DBN* [19]	200 - 200	-120.46
DBM* [17]	200 - 200	-118.73
LRBN	200	-102.21
LRBN	200 - 200	-97.49

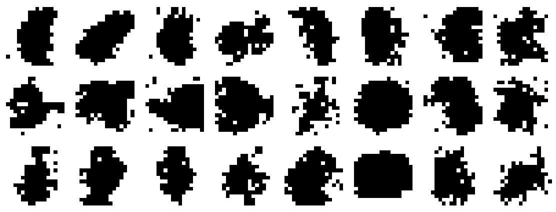


Figure 4: Random samples from the generative model on Caltech 101 Silhouettes dataset.

6.4 OCR letters dataset

The last experiment is performed on the OCR letters dataset, which contains 42,152 training images and 10,000 testing images of English letters. The images have the dimensionality of 16×8.

The reconstruction error is reported in Table 5. The proposed method shows superior performance compared to all the competing methods. The average reconstruction error on the test set is 5.95 pixels, which is at least 17 pixels better than the other methods.

The test data log-probability is reported in Table 6. Our model obtains a variational lower bound of -35.02, which outperforms the variational Bayes learning method, and is slightly worse than DBM [18], which has 100 times more parameters. Samples from the LRBN are shown in Fig. 5. We display the samples of letter 'g'. For the same letter, the learned model is able to capture the different handwriting styles, while preserving the key information.

Table 5: Average reconstruction errors of different methods on OCR letters dataset.

Method	DIM	Recon Error
NVIL* [13]	200 - 200	14.79
RBM* [4]	200	16.83
DBN* [19]	200 - 200	12.47
DBM* [17]	200 - 200	11.14
LRBN	200 - 200	2.03

Table 6: Test data log-probability on OCR letters dataset.

Method	DIM	Log-prob
VB [5]	200	-48.20
VB [5]	200 - 200	-47.84
DBN* [19]	200 - 200	40.75
DBM [18]	2000 - 2000	-34.24
LRBN	200	-39.48
LRBN	200 - 200	-35.02

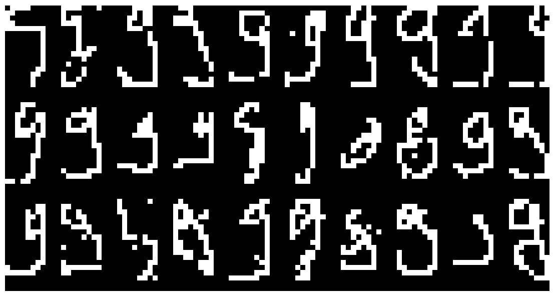


Figure 5: Random samples from the generative model on OCR letters dataset.

7 Conclusion

In this work, we introduce a directed deep model based on the latent regression Bayesian network to explicitly capture the dependencies among the latent variables for data representation. We introduce an efficient inference method based on pseudo-likelihood and coordinate ascent. A hard EM learning method is proposed for efficient parameter learning. The proposed inference method solve the inference intractability, while preserving the dependencies among latent variables. We theoretically and empirically compare different models and learning methods. We point out that the latent variables in regression Bayesian network have strong dependencies, which can better explain the patterns in the input layer. Experiments on benchmark datasets shows the proposed model significantly outperforms the existing models in data reconstruction and achieves comparable performance for data representation.

References

[1] Yoshua Bengio, Li Yao, and Kyunghyun Cho. Bounding the test log-likelihood of generative models. In *International Conference on Learning Representations (Conference Track)*, 2014.

[2] Julian Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 259–302, 1986.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[4] K Cho, Tapani Raiko, and Alexander Ilin. Enhanced gradient for training restricted boltzmann machines. *Neural computation*, 25(3):805–831, 2013.

[5] Zhe Gan, Ricardo Henao, David Carlson, and Lawrence Carin. Learning deep sigmoid belief networks with data augmentation. *International Conference on Artificial Intelligence and Statistics*, 2015.

[6] Zoubin Ghahramani and Geoffrey E Hinton. The em algorithm for mixtures of factor analyzers. Technical report, Technical Report CRG-TR-96-1, University of Toronto, 1996.

[7] Karol Gregor, Andriy Mnih, and Daan Wierstra. Deep autoregressive networks. *In Proceedings of the 31st International Conference on Machine Learning*, 2014.

[8] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[9] Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214): 1158–1161, 1995.

[10] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[11] Thomas Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 289–296. Morgan Kaufmann Publishers Inc., 1999.

[12] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *In Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[13] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *In Proceedings of the 31st International Conference on Machine Learning*, 2014.

[14] Radford M Neal. Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113, 1992.

[15] Ankit B Patel, Tan Nguyen, and Richard G Baraniuk. A probabilistic theory of deep learning. *arXiv preprint arXiv:1504.00641*, 2015.

[16] Danilo J Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1278–1286, 2014.

[17] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009.

[18] Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 693–700, 2010.

[19] Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th international conference on Machine learning*, pages 872–879. ACM, 2008.

[20] Lawrence K Saul, Tommi Jaakkola, and Michael I Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4(61):76, 1996.

[21] Yichuan Tang, Geoffrey E Hinton, and Ruslan Salakhutdinov. Deep mixtures of factor analysers. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 505–512, 2012.

[22] Aaron van den Oord and Benjamin Schrauwen. Factoring variations in natural images with deep gaussian mixture models. In *Advances in Neural Information Processing Systems*, pages 3518–3526, 2014.